Chapitre 05 LANGAGE MACHINE

Introduction

L'exécution d'un programme consiste à donner à la machine une séquence d'instructions directement interprétable. Les premiers programmes été écris en binaire (ils sont difficiles et risque d'erreurs). Pour comprendre une séquence de bits, il convenait de créer une table décrivant toutes les opérations possibles et leurs représentations binaires, c'est l'époque du langage machine. Par la suite, les programmes ont été écrits en donnant directement les noms abrégés des opérations, on les appelait les codes mnémoniques (ADD, DIV, SUB, INC …). Chaque commande ou code opération (binaire) est représenté par un mnémonique. Par exemple, la commande d'addition s'écrit ADD en abrégé et correspond au code opération 0001 0101 par exemple. Les adresses des instructions et des variables pouvaient aussi être donné sous la forme symbolique. Le langage qui utilise des instructions écrites en mnémonique est appelé langage d'assemblage ou langage assembleur.

Le langage assembleur est classé dans la catégorie des langages de bas niveau ; c'est un langage étroitement lié au processeur ou microprocesseur et aux circuits d'un ordinateur, cela à l'opposé des langages de haut niveau, qui sont indépendants du processeur. Un programme développé en assembleur est exécuté bien plus rapidement que s'il avait été développé en langage évolué. Il occupe aussi beaucoup moins de place en mémoire car il est plus compact.

Un programme est décomposé en actions élémentaires, en instructions de base du processeur, l'assembleur mène toutefois à des programmes bien plus longs à écrire qu'en langages évolués ; plus longs aussi à vérifier et à corriger. En outre un programme écrit pour un processeur donné ne fonctionnera pas avec un autre processeur différent ou d'une famille différente : il n'est pas portable.

Pour pouvoir exécuter un programme rédigeait à l'aide de mnémoniques en langage assembleur par l'ordinateur, ce programme doit être converti en langage machine (binaire); ce fût réalisé par l'assembleur, c'est le traducteur qui convertit les programmes écrits en langage d'assemblage en langage machine. Le langage d'assemblage permet d'exploiter au maximum les ressources de la machine. En langage d'assemblage, le programmeur peut avoir accès à toutes les caractéristiques de la machine cible (non possible dans les langages évolués).

Exemple:

- lire les états des interrupteurs de la console.
- accès aux différents registres.
- diagnostique d'erreur (soft et hard).

Fig.01. Principe de préparation d'un programme en assembleur.

1. Instruction machine

Le langage machine se compose d'instructions binaires telles qu'on les trouve dans la mémoire au moment de l'exécution du programme. Le langage d'assemblage n'est qu'une variable symbolique du langage machine, il a le même jeu d'instructions. Il est propre à chaque type machine. On a un langage d'assemblage pur lorsque chaque instruction de ce langage produit exactement une instruction en langage machine.

1.1. Format d'une instruction: Bien que la structure des instructions d'un langage d'assembleur reflète l'architecture de la machine, on peut dégager quelques grandes caractéristiques des instructions machine.

Exemple: pour calculer N :=I+J+K

a) IBM 370

	champ étiquette	champ opération	champ opérande	commentaires
Instructions	formule	L	1, I	charger I dans le registre 1
		A	1, J	additionner J au registre 1
		A	1, K	additionner K au registre 1
		ST	1, N	stocker le contenue du registre 1 dans l'adresse N
Pseudo instructions (directives)	I	DC	'5'	
	J	DC	'8'	
	K	DC	'10'	
	N	DC	'0'	

b) PDP 11

	champ étiquette	champ opération	champ opérande	commentaires
Instructions	form :	MOV	I, R_1	charger I dans le registre 1
		ADD	J, R_1	additionner J au registre 1
		ADD	K, R_1	additionner K au registre 1
		MOV	R_1, N	stocker le continue du registre 1 dans l'adresse N
Pseudo instructions (directives)	I:	5		
	J:	8		
	K:	10		
	N:	0		

Une instruction en langage assembleur est devisée en plusieurs champs:

étiquette	Code opération (mnémonique)	Opérandes

Fig.02. Format d'une instruction machine

Les différents champs sont séparés par un ou plusieurs espaces, le nombre des opérandes du 3ème champ varie d'une machine à l'autre de 0 a 3. Après ce champ, on peut ajouter des commentaires.

1.1.1. Champ étiquette: On utilise les étiquettes pour donner des noms symboliques à certaines adresses mémoires, elles sont indispensables lorsque l'on désire effectuer des branchements aux instructions qu'elles préfixent. Elles permettent aussi, dans les directives (pseudo instructions) d'allocation mémoire, d'accéder en suite aux données par un nom symbolique.

1.1.2 Champ opération: Il contient soit un mnémonique dans le cas de la représentation d'une instruction machine, soit une directive si l'instruction est un ordre donné à l'assembleur, le choix des mnémoniques pour la même instruction varie d'un constructeur à un autre.

Exemple: L ⟶ IBM 370
MOV ⟶ PDP 11

Chaque instruction correspond pour l'unité centrale (UC) à l'exécution d'un travail. Le champ le plus important est le champ opération car il indique l'opération à l'UC l'opération à entreprendre. Les autre champs contiennent ou spécifient les données ou les opérandes nécessaires à l'exécution de l'instruction.

Exemple: Une instruction de comparaison.
- le code opération indique le type de comparaison à réaliser (<, >, = …etc).
- les autres champs précisent les opérandes à comparer.

La façon d'obtenir les opérandes d'une instruction s'appelle l'adressage. Sur certaines machines, toutes les instructions sont de la même longueur, sur d'autres machines on distingue deux ou trois tailles d'instructions selon le type d'adressage et le type d'instruction.

a)

Code opération

b)

Code opération	adresse

c)

Code opération	Adresse 1	Adresse 2

Fig.03. Format typique des instructions

1.2. Critères d'évaluation du format des instructions

Lors de la conception de la couche machine traditionnelle, la spécification des formats des instructions a une très grande importance :

1. Les instructions les plus courtes sont les plus efficaces au niveau des performances d'exécution et de l'occupation mémoire des programmes. La mémoire est caractérisée par sa capacité de stockage et par sa vitesse (débit) de transfert des informations (bits/s). Une mémoire de débit ***t*** bits/s et de longueur d'instructions de ***nb*** bits $\Rightarrow$ ***débit max = t/nb*** instructions/seconde.

2. La vitesse de traitement des instructions est un paramètre propre a l'UC, elle dépend plus au moins fortement de la taille des instructions. Si le temps de traitement est supérieur au temps de recherche $\Rightarrow$ pas d'influence sur le temps global, sinon si le processeur est très rapide $\Rightarrow$la mémoire ralentit le processus d'exécution des instructions.

3. L'évaluation de la largeur des instructions (code opération en bits) afin d'exprimer toute les opérations envisagées.

4. L'évaluation de la taille du mot machine est importante car il est strictement lié à la taille des instructions. Pour être optimal, la taille du mot machine doit être un multiple du nombre de bits de caractères manipulés. Si la longueur d'un caractère est de ***k*** bits $\Rightarrow$ le mot mémoire à une longueur ***k***, ***2k, 3k***…. Pour éviter le gaspillage de la mémoire centrale, les instructions doivent être formées d'un nombre entier de caractères ou mots mémoire.

5. La largeur des champs adresse et opérandes.

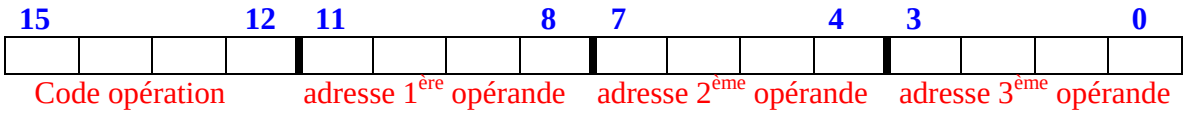
1.2. Code opération expansif

Considérons un mot de *(k+n)* bits de longueur, dont *k* bits pour le code opération (2^k opérations possibles) et *n* bits pour le champ adresse (2^n mots mémoires adressables). D'une autre manière, cette instruction de *(k+n)* bits pourrait se décomposer en *(k-1)* bits de code opération et de *(n+1)* bits d'adresse, ce qui se traduit par moitié moins d'opérations et deux fois plus d'espace mémoire adressable et le contraire est juste. On voit qu'il est possible, partant d'une configuration donnée en nombre de bits, on peut déterminer différents formats d'instructions et cela pour la même machine: c'est la notion du code opération expansif mis en œuvre sur certaines machines et notamment sur le PDP 11.

Code opération	Adresse	
k bits	n bits	$\Rightarrow 2^k$ opérations et 2^n adresses possibles.
k-1 bits	n+1 bits	$\Rightarrow 2^{k-1}$ opérations et 2^{n+1} adresses possibles.
k-2 bits	n+2 bits	$\Rightarrow 2^{k-2}$ opérations et 2^{n+2} adresses possibles.

Fig.O4. Code operation expansif

Exemple : Soit une instruction codée sur 16 bits definit comme suit :



Cette instruction, composée d'un code opération sur 4 bits et trois champs adresse, permet de définir 16 instructions à trois opérandes. Avec ce format d'instruction, on peut avoir le jeu d'instructions en utilisant la notion de code opération expansif:

15 instructions à trois opérandes dont le code opération (bits 12 à 15) varie de 0000 à 1110 (de 0 à 14). La configuration 1111 définit un nouveau code opération formé par expansion avec les bits 8 à 11, les bits de 0 à 3 et de 4 à 7 définissent deux champs adresse. On obtient de nouvelles instructions dont le code opération est constitué des bits 8 à 15 (dont les bits 12 à 15 valent tous 1) et les bits 8 à 11 évoluent de 0 à 13 (soit 14 instructions à deux opérandes). Les configurations 1110 et 1111 des bits 8 à 11 sont interprétées différemment. Ces deux configurations définissent un autre code opération formé par expansion avec les bits 4 à 7, les bits de 0 à 3 définissent le champ adresse, on obtient ainsi un nouveau groupe d'instructions dont le code opération est formé des bits 3 à 15 (soit 32 opérations possibles) , on prend 31 instructions à une seule opérande et la configuration 111111111111 est interprétée différemment. Cette dernière configuration définit par expansion avec les bits 0 à 3 un nouveau code opération sur 16 bits, on obtient ainsi 16 instructions sans opérandes (sans champs adresse)

Code opération sur 4 bits :

0000 xxxx yyyy zzzz
0001 xxxx yyyy zzzz
:
:
:
1101 xxxx yyyy zzzz
1110 xxxx yyyy zzzz

} 15 instructions à 3 opérandes

Code opération sur 8 bits :

1111 0000 xxxx yyyy
1111 0001 xxxx yyyy
:
:
:
1111 1101 xxxx yyyy

} 14 instructions à 2 opérandes

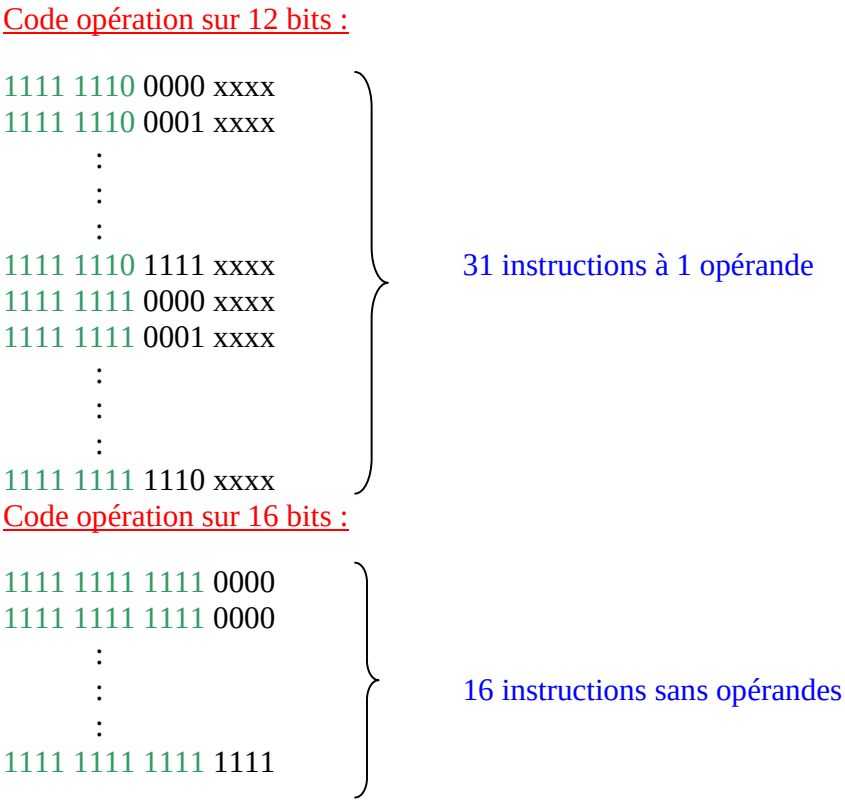


Fig.05. Exemple de code opération expansif

2. Modes d'adressage

On peut classer les instructions d'une machine selon le nombre de champs adresse qu'elles contiennent. Les champs adresses peuvent représenter (référencier) des mots mémoires ainsi que des registres (mémoire à accès rapide). On distingue généralement des instructions à une, deux voire trois champs adresse. Sur un nombre de machines, les opérations sont réalisées avec des instructions à une seule adresse d'opérande, dans ce cas un registre particulier (ACC) contient le second opérande. Sur ces machines le champ adresse correspond le plus souvent à l'adresse d'un mot mémoire qui contient l'opérande. Ainsi, une instruction d'addition à une seule adresse du mot mémoire m est notée :

ACC := ACC + M[m]

Une instruction à deux adresses dispose d'une adresse source et d'une adresse destination, la source est additionnée à la destination.

Destination := destination + source

Les instructions à trois adresses déterminent deux sources et une destination, les sources sont additionnées puis le résultat est rangé à la destination.

Destination := source1 + source2

Pour permettre à l'UC d'accéder a l'opérande, on peut spécifier l'adresse effective directement dans le champ adresse ou utiliser d'autres possibilités appelées modes d'adressage des instructions.

2.1. L'adressage immédiat

La façon la plus simple consiste à inscrire l'opérande dans la champ adresse de l'instruction plutôt qu'une adresse ou une information précisant sa localisation. On appel cette opérande « opérande immédiat» ou « valeur immédiate» car elle est automatiquement chargée de la mémoire dans l'UC lors du cycle de recherche de l'instruction, c'est le mode d'adressage immédiat.

Ce mode offre l'avantage d'éliminer l'accès supplémentaire à la mémoire pour obtenir l'opérande concerné. Il a par contre les inconvénients suivants:

a. la valeur de l'opérande est limitée par le nombre de bits du champ adresse de l'opérande.

b. La valeur de l'opérande est fixée une fois pour toute lors du codage de l'instruction et ne peut plus évoluée.

Le PDP 11 dispose de ce mode d'adressage où l'opérande immédiat est codé sur 16 bits en complément à 2.

Code opération	«Opérande immédiat»	Champ adresse 2

MOV	100	Champ adresse 2

Fig.06. Mode d'adressage immédiat

2.2. L'adressage direct

Dans ce mode d'adressage, l'adresse effective de l'opérande est inscrite directement dans le champ adresse de l'instruction.

Code opération	«Adresse de l'opérande»	Champ adresse 2

MOV	**200**	**100**

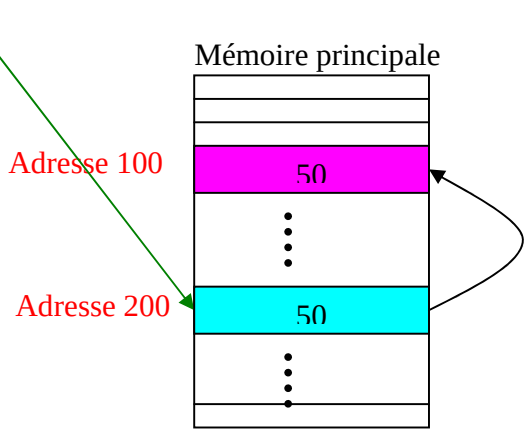


Fig.07. Mode d'adressage Direct

2.3. L'adressage par registre

Il est conceptuellement le même que le direct. Cependant, dans ce mode le champ adresse de l'instruction fait référence à un registre interne de l'UC qui contient l'opérande concerné. Il faut différencier entre les espaces d'adressage (registres et mémoire), pour cela on peut envisager que l'adresse est décomposée en deux parties: une partie indiquant la nature de l'espace adresse et le reste du champ pour préciser l'adresse de l'élément dans l'espace choisi. Comme le nombre de registres est largement inférieur au nombre de mots mémoire, l'adresse spécifiant un registre est plus courte que celle d'un mot mémoire. Cela suppose des formats d'instructions différents pour adresser les registres ou la mémoire. On peut envisager que la différentiation des formats se fasse directement dans le code opération de l'instruction.

Code opération	Mode d'@	L'@ du registre	Champ adresse 2

MOV	**Mode @ registre**	**R1**	**Mode @ direct**	**100**

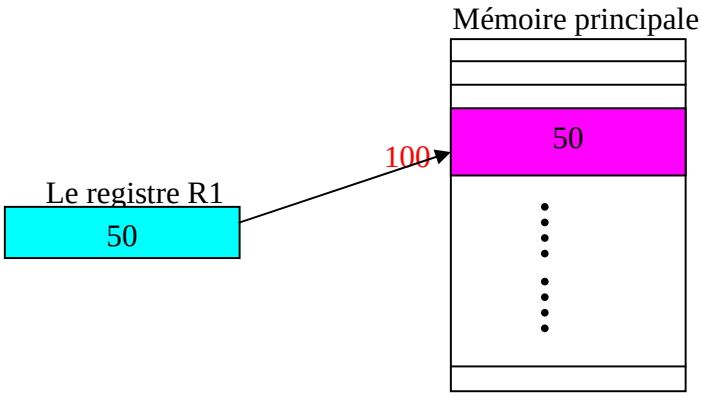


Fig.08. Mode d'adressage par registre

2.4. L'adressage indirect

Ce mode consiste à inscrire dans le champ adresse de l'instruction non pas l'adresse de l'opérande mais l'adresse d'un mot mémoire ou d'un registre qui contient l'adresse effective de l'opérande.

En premier lieu, le contenu du mot mémoire d'adresse 100 est chargé dans un registre interne de l'UC. Cette valeur 200 n'est pas chargée dans le registre R1, sinon on réaliserait un chargement en mode d'adressage direct. Au lieu de cela, le contenu du mot mémoire situé à l'adresse 200 qui est chargé en R1. La valeur 200 qui se trouve à l'adresse 100 n'est pas une opérande mais une adresse qui pointe vers un opérande ; on l'appelle souvent pointeur.

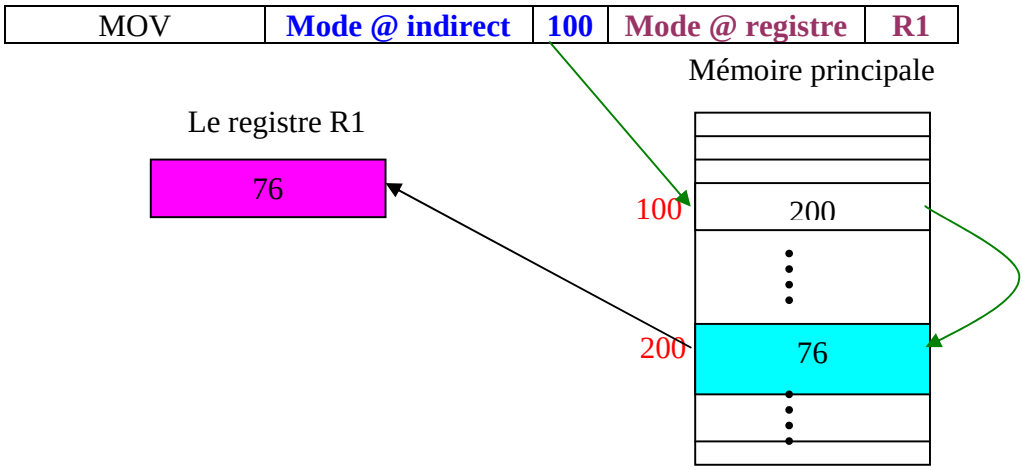


Fig.09. Mode d'adressage par registre

2.5. L'adressage indexé

De nombreux algorithmes nécessitent la réalisation successive d'opérations portant sur un ensemble de données qui se trouvent en mémoire à des adresses consécutives.

Exemple: Considérons un bloc de *n* mots situés aux adresses *A, A+1, …, A+n-1* que l'on veut déplacer aux adresses *B, B+1, …, B+n-1*.

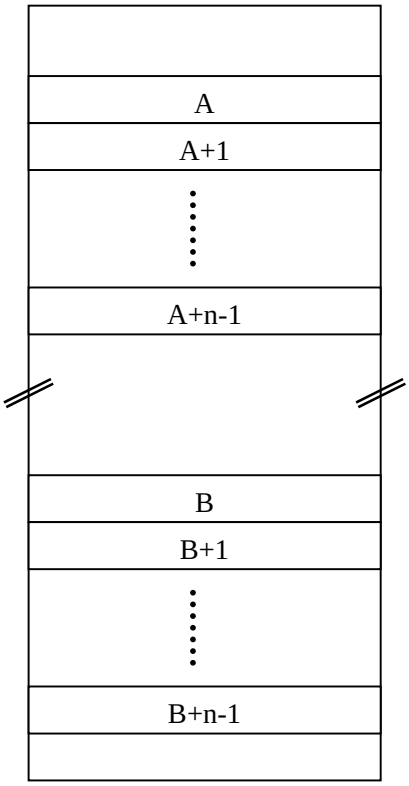


Fig.10. Transfert du bloc mémoire d'adresse A à l'adresse B

La machine dispose de l'instruction *MOV A,B* qui transfert le mot d'adresse A dans celui d'adresse B, une fois cette opération est réalisée on modifie l'instruction pour effectuer le transfert suivant *MOV A+1,B+1*.

On répète l'exécution de cette instruction après modification de l'adresse jusqu'à ce que les *n* mots soient copiés de la zone A vers la zone B. cette suite d'instructions identiques constitue une lourdeur du programme.

Ce problème de copie peut être résolut en utilisant le mode d'adressage indirect. En effet, un registre ou un mot est chargé avec la valeur *A* et un autre avec la valeur *B*. l'instruction *MOV* en mode d'adressage indirect utilise les deux pointeurs qui sont incrémentés après chaque utilisation de un jusqu'à ce que les *n* mots soient copiés de la zone *A* vers la zone *B*.

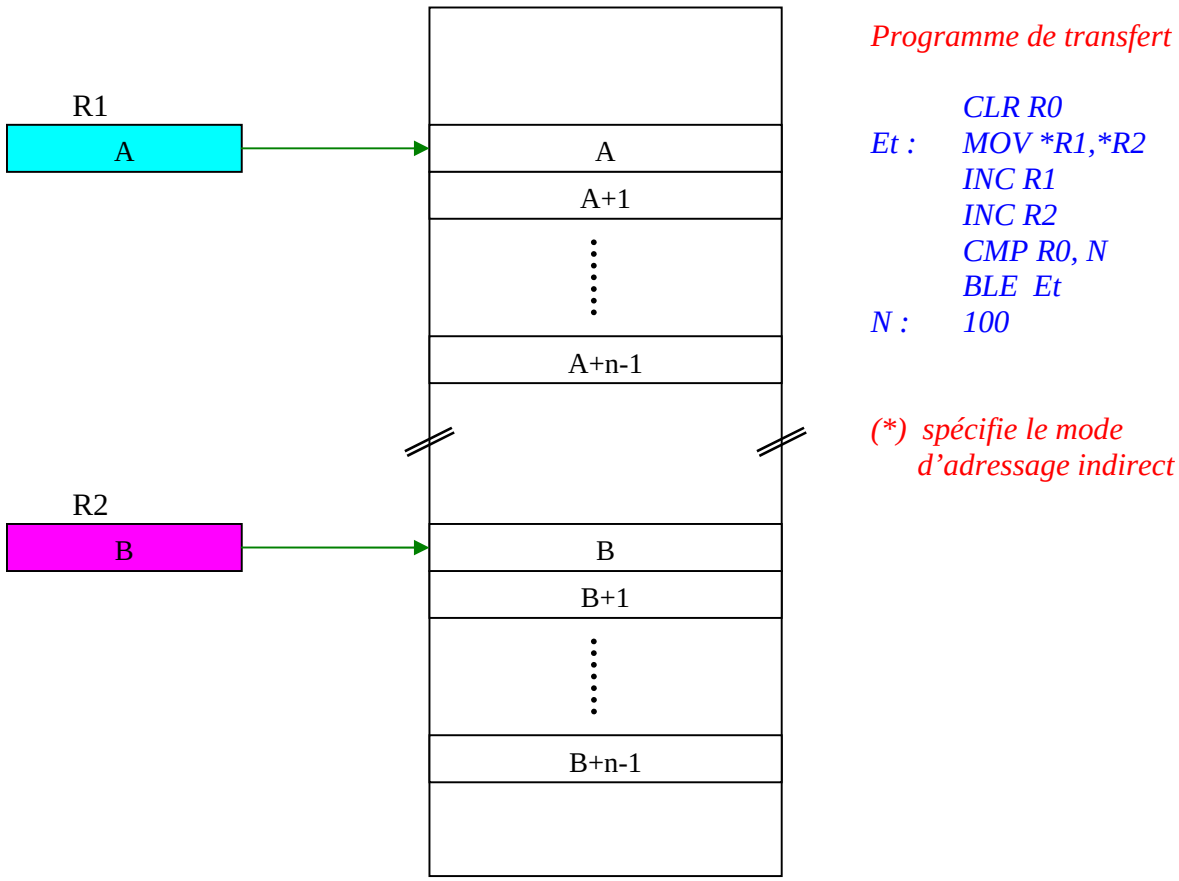


Fig.11. Utilisation du mode d'adressage indirect pour le transfert du bloc mémoire d'adresse A à l'adresse B

Une autre méthode consiste à utiliser un registre particulier appelé registre index. Dans ce cas, la partie adresse comprend le nom du registre index et une constante d'indexation. L'adresse de l'opérande est obtenue par addition du registre index avec la constante ; c'est le mode d'adressage indexé.

Exemple: Supposons que les deux adresses *A* et *B* sont indexés avec un même registre d'index qui contient la valeur *k*. L'instruction *MOV A,B* réalise le transfert du mot d'adresse *A+k* vers l'adresse *B+k*. Pour réaliser le transfert de *A* vers *B*, il suffit d'initialiser le contenue du registre d'index à 0 et l'incrémenter après chaque exécution de l'instruction *MOV*.

Remarque: L' incrémentation d'un registre est plus rapide que celle d'un mot mémoire, d'où la préférence de l'adressage indexée dans ce cas. L'indexation est également utilisée pour adresser une zone ou un champ situé à une certaine distance du début d'une structure de données.

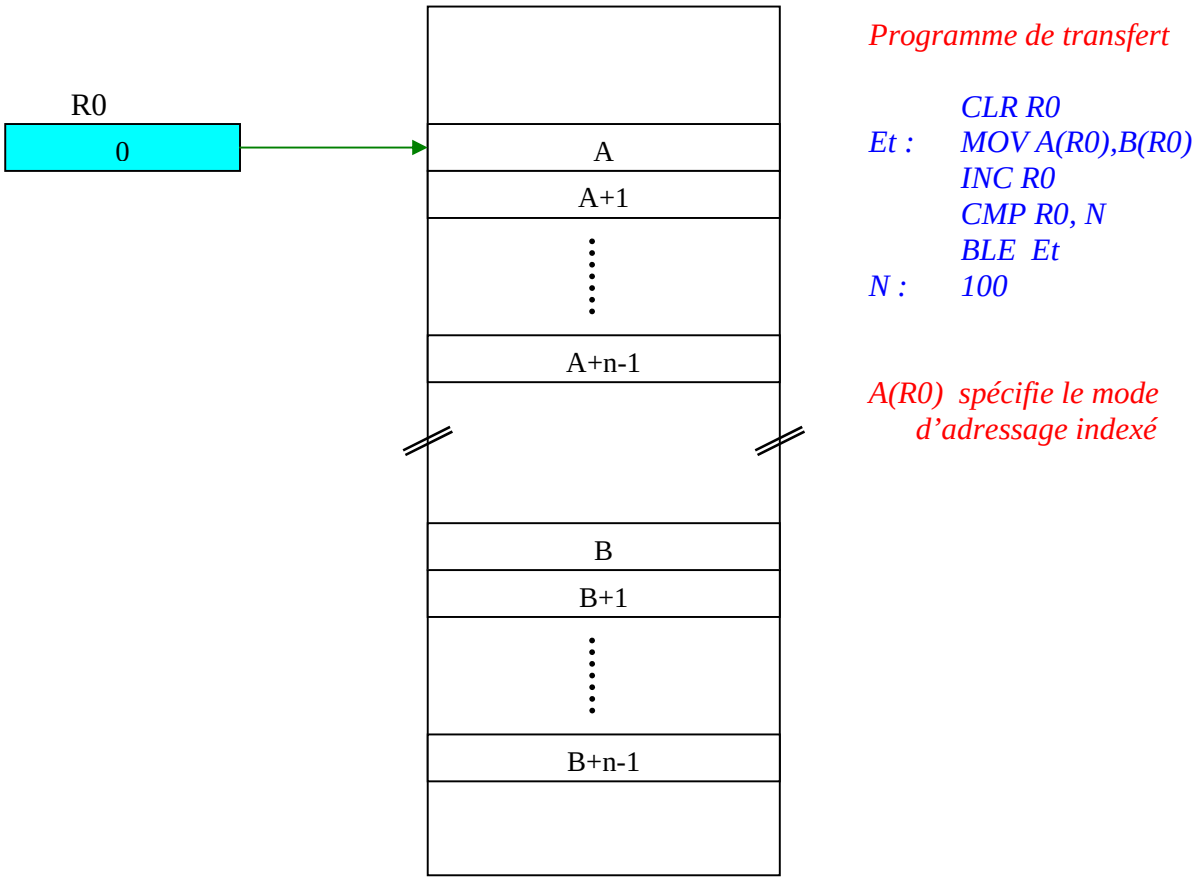


Fig.12. Utilisation du mode d'adressage indexé pour le transfert du bloc mémoire d'adresse A à l'adresse B

Toutes les instructions du PDP11 qui se réfèrent à la mémoire centrale peuvent être utilisées en mode d'adressage indexé. Le registre index pouvant être l'un des huit registres généraux de cette machine. De plus les instructions disposant de deux champs adresse mémoire peuvent associer (ou non) des index différents à chaque adresse. L'opération d'incrémentation ou décrémentation du registre index peut s'effectuer automatiquement. Le PDP11 permet l'auto-indexation de n'importe lequel de ces registres généraux.

2.6. L'adressage par registre de base

Le plus souvent, les données comme les programmes sont rangés en mémoire dans des régions ou zones très localisées. Certaines machines exploitent cette caractéristique en réduisant le nombre de bits des adresses mémoires. La longueur en bits des adresses mémoires peut être réduite par l'utilisation d'un certain nombre de registres de base, dont chacun contient un pointeur vers une zone d'informations en mémoire centrale. Une adresse est composée dans ce cas de deux parties : le nom du registre de base et la valeur entière appelée déplacement.

L'adresse effective de l'opérande est obtenue par addition du déplacement au contenu du registre de base. Un registre de base doit comprendre un nombre de bits suffisant pour pointer sur n'importe quelle adresse mémoire.

Exemple 01: une machine disposant de n registres et d'un déplacement codé sur k bits $\Rightarrow$ on peut obtenir n zones mémoire de 2^k adresse (mots mémoire) chacune.

Exemple 02: une machine qui dispose de quatre registres de base de 32 bits et d'un déplacement codé sur 12 bits $\Rightarrow$ quatre zones mémoire chacune de 2^{12} adresses (4096 mots mémoires) voir figure *fig.09.*, ainsi un champ adresse est constitué de 14 bits : 2 bits pour le nom du registre de base, 12 bits pour le déplacement.

Dans les deux techniques d'adressages indexé et par registre de base; on réalise l'addition du contenu du registre avec une constante située dans le corps de l'instruction pour obtenir l'adresse effective de l'opérande. Conceptuellement, il n'y a aucune différence entre les deux, cependant lorsque le nombre de bits du déplacement est suffisant pour adresser la mémoire entière, on parle alors d'adressage indexé, dans le cas contraire; Il s'agit d'adressage par registre de base.

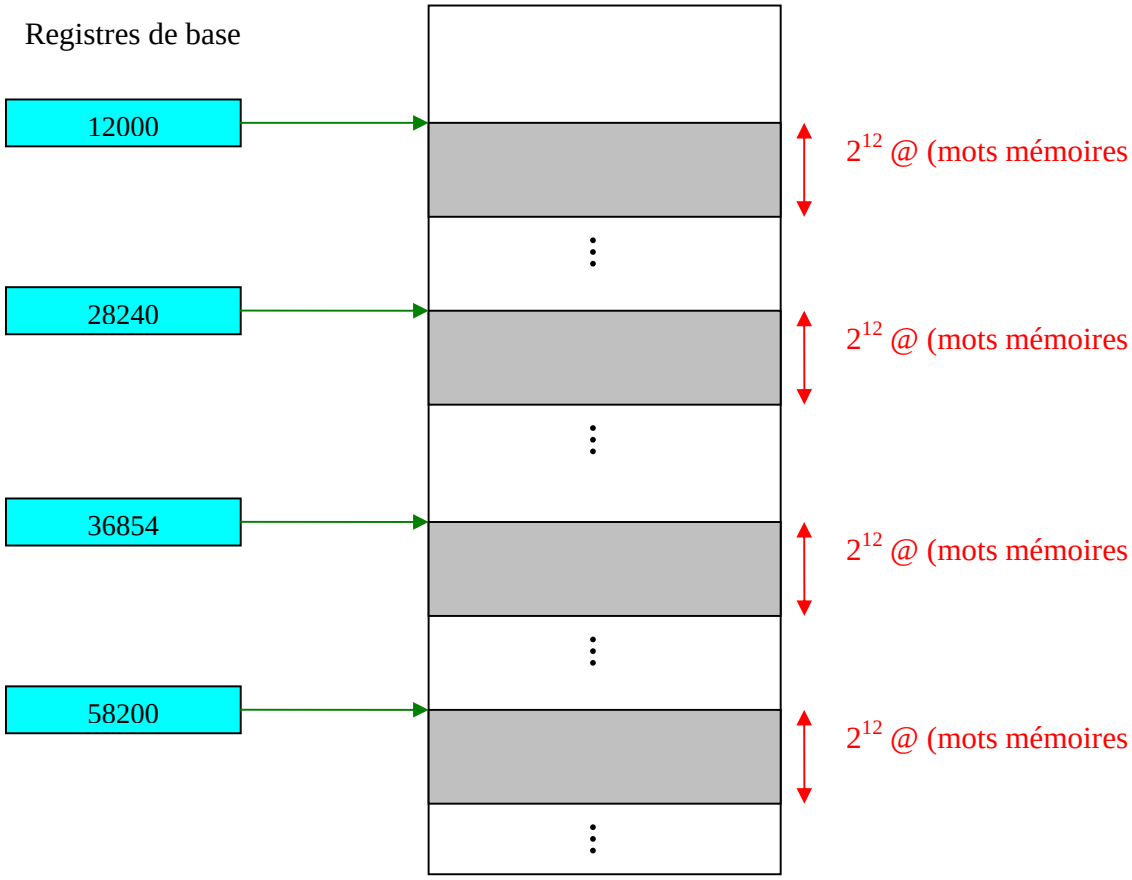


Fig.13. Utilisation du mode d'adressage par registre de base.

2.7. **L'adressage par pile**

Il est important que les instructions soient les plus courtes possibles afin d'économiser l'espace mémoire et le temps de traitement par l'UC. L'idée est de réduire le nombre de bits du champs adresse à 0. Par conséquence, on obtient des instruction sans opérandes (adresse ou champ adresse), cette situation consiste à organiser la machine autour d'un structure de pile. Une pile est constituée d'un ensemble d'informations (mots, caractères…etc) stockées séquentiellement en mémoire. Le premier élément enregistré dans la pile est placé à la base de la pile, le dernier élément entré dans la pile se trouve à son sommet. En association avec une pile, une machine dispose d'un pointeur de pile (un registre particulier ou un registre général) qui contient l'adresse du sommet pile a tout moment.

L'instruction PUSH charge automatiquement une donnée (contenue dans un mot mémoire ou un registre) et incrémente le pointeur de pile. L'instruction POP extrait la donnée du sommet pile, l'enregistre en mémoire ou dans un registre et décrémente le pointeur de pile.

Les instructions sans adresses sont utilisées par la machine conjointement avec une structure de pile. Cette forme particulière d'adressage spécifie que les deux opérandes de l'instruction se trouvent dans la pile et qu'ils seront extraits (2 POP) l'un après l'autre au fur et a mesure de l'exécution de l'instruction, par exemple la multiplication, un et logique …etc. Le résultat du traitement est lui aussi rangé dans la pile (PUSH).

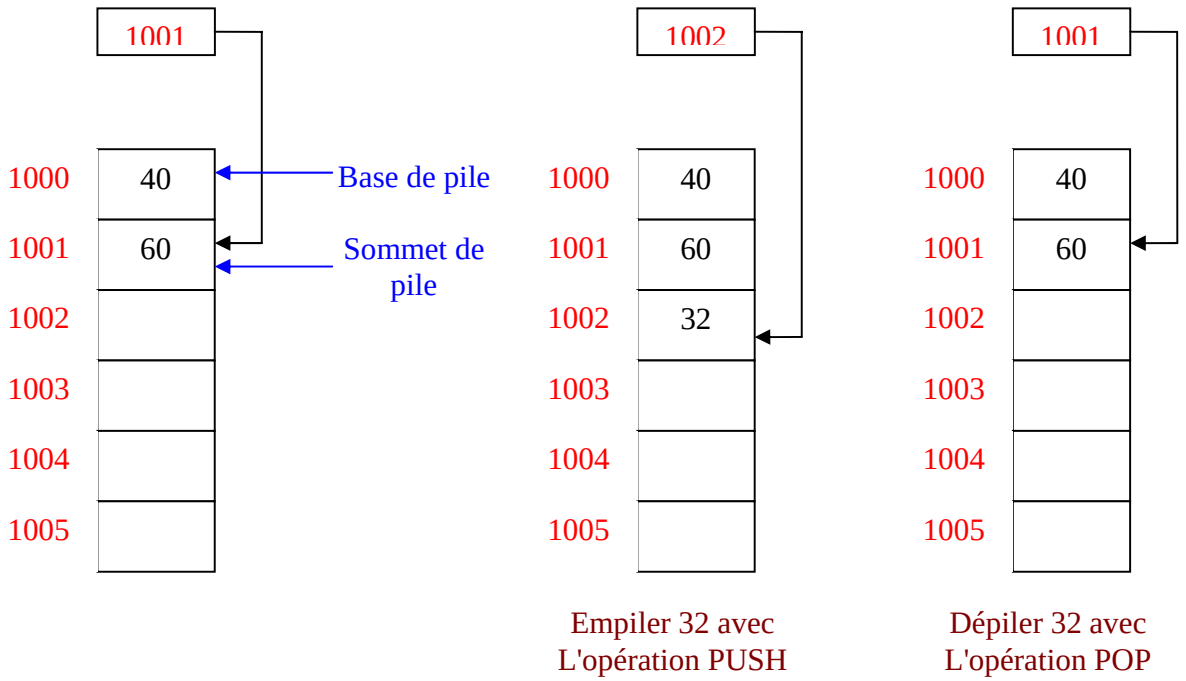


Fig.14. Exemple du mode d'adressage par Pile.

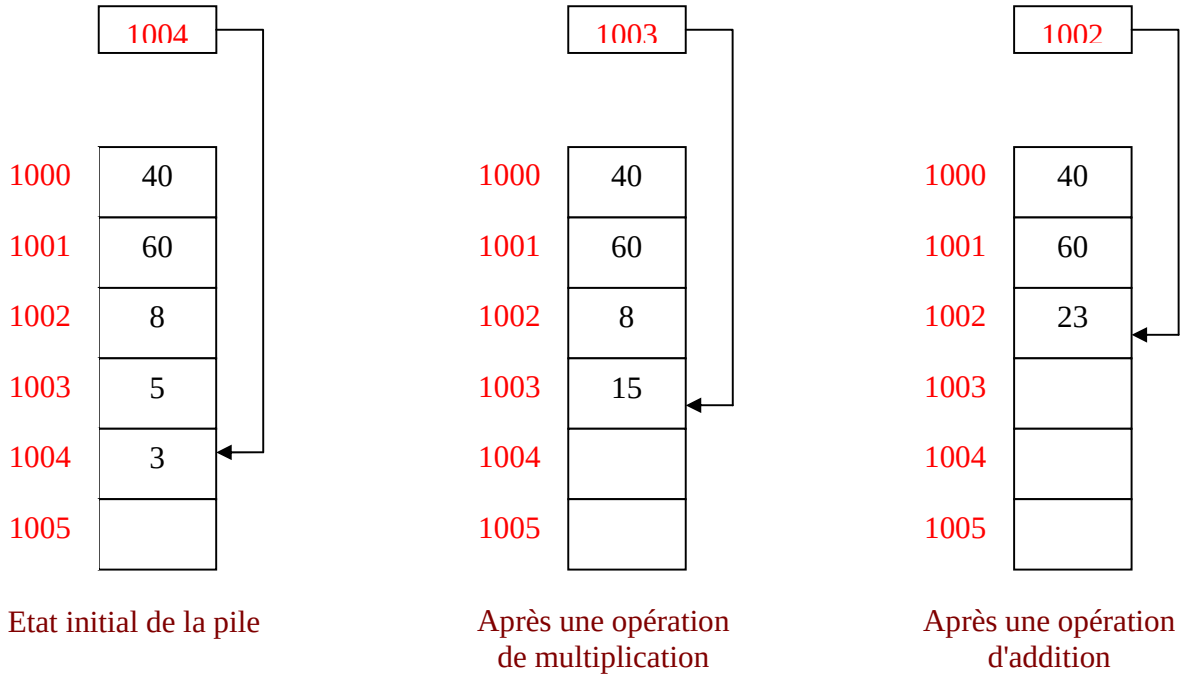


Fig.15. Exemple de l'utilisation d'une pour le calcul numérique.